Variability and Complexity in Software Design – Towards Quality through Modeling and Testing

Matthias Galster
University of Canterbury
Christchurch, New Zealand
mgalster@ieee.org

Danny Weyns
KU Leuven, Belgium
Linnaeus University, Sweden
danny.weyns@kuleuven.be

Michael Goedicke
University of Duisburg-Essen
Essen, Germany
michael.goedicke@s3.uni-due.de

Uwe Zdun
University of Vienna
Austria, Vienna
uwe.zdun@univie.ac.at

Jácome Cunha
NOVA LINCS, DI, FCT, NOVA University of Lisbon
Lisbon, Portugal
jacome@fct.unl.pt

Jaime Chavarriaga
Universidad de los Andes
Bogotá, Colombia
ja.chavarriaga@uniandes.edu.co

ABSTRACT
Today's software systems must accommodate a wide range of usage and deployment scenarios. The increasing size and heterogeneity of software-intensive systems, dynamic and critical operating conditions, fast moving and highly competitive markets, and increasingly powerful and versatile hardware makes it more and more difficult to handle the additional complexity in design caused by variability. This paper reports results of the Second International Workshop on Variability and Complexity in Software Design. It also outlines directions the field might move in the future.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures

General Terms
Management, Documentation, Design.

Keywords
Variability, complexity, software design.

1. INTRODUCTION
1.1 Overview
VACE 2017 addressed software engineering challenges related to requirements, design, implementation, evaluation, deployment, operation and maintenance "variability-intensive" software and the complexity in the design of those systems. VACE 2017 embraced the notion of "variability-intensive" which includes any type of system that needs to accommodate diverse application and deployment scenarios (e.g., due to variations in users and user needs, dynamics in the availability of resources or external services, market segments, customer profiles, different emphases in different phases of the software development process, or variation in hardware resources). Examples of such "variability-intensive" systems include self-adaptive systems, configurable or customizable single systems, open platforms, context-aware mobile apps, plug-ins of web browsers, service-based and cloud-based systems, IoT and cyber-physical systems. Such systems can range from small-scale embedded systems to large-scale enterprise systems to ultra-large systems of systems.

Variability became a concern of most modern software systems to accommodate different deployment and usage scenarios since today's stakeholders and software users expect flexibility in many dimensions, e.g., features, location and resource awareness, fault tolerance, and energy consumption of mobile devices. Therefore, variability needs to be faced in mainstream and pervasive software and is not limited to "traditional" fields such as software product lines and families anymore. A trend in the next decade will be managing variability in a non-product line context and under open-world assumptions. Also, currently separated research communities will need to cooperate closer.

In software engineering a design space comprises the set of possible design options and design parameters that could potentially meet a specific software system's requirements. Designing for variability means considering highly diverse stakeholders and extremely large design spaces. This complexity of the design space because of variability becomes even more challenging in the light of current trends:

- Increasing size and heterogeneity of software-intensive systems (e.g., software ecosystems, cyber-physical system, systems of systems, ultra-large scale systems)
- New and emerging application domains (e.g., unmanned aerial vehicles and self-driving cars, smart health apps and sensor-based systems, large-scale surveillance systems, software-defined networking)
- Dynamic and critical operating conditions under which software must function (e.g., available resources and services, disaster monitoring and response systems)
- Fast moving and highly competitive markets (e.g., gaming, mobile) as well as more powerful/versatile hardware (e.g., Raspberry Pi)

Consequently, designing for, implementing, operating and maintaining variability in software systems not only affects the software's functionality and quality (i.e., *what* do we build), e.g., systems for "continuous configuration management" from compilation to deployment to runtime [5]. It also affects the development process (i.e., *how* we build it), e.g., systematic quality assurance and validation despite a potentially large and complex design and solution space. Moreover, in some new consumer domains of critical systems, e.g., autonomous and unmanned vehicles, research is only slowly catching up with industry trends and needs. Such systems soon become an integral part of many industries, including construction, agriculture, emergency responder support, etc. Once this happens, practices need to be in place to help develop such systems. Additionally, successful companies are innovative companies that target new market opportunities, independent of solutions or ideas that currently exist. On the other hand, the time to market can make the difference between product success and failure. This highlights the need for more "light-

weight" approaches to variability-intensive systems, which balance the need for innovation but also consider reducing development effort, even for innovative products. New product models for variability-intensive systems could help manage system growth over time and offer opportunities for innovation throughout development. Finally, there is a need-supply gap in engineering capability (processes, practices but also in skills and workforce). Issues described above have created new engineering needs in which old work practices do not apply.

1.2 Workshop History

The Second International Workshop on Variability and Complexity in Software Design (VACE) was held in conjunction with the International Conference on Software Engineering (ICSE) in Buenos Aires, Argentina. The workshop website can be found at http://vaquita-workshop.org/vace2017/. Previously, the first edition of VACE was collocated with ICSE 2016 in Austin, Texas [7].

VACE is an evolution of the VARSA workshop series (International Workshop on Variability in Software Architecture) held at WICSA in 2011 [3], 2012 [6], and 2014 [4], and VAQUITA (Workshop on Variability for Qualities in Software Architecture) held at ECSA 2015. Evolving these two workshops into one ICSE workshop broadened the community beyond software architecture to reach an audience with a much broader and diverse background and expertise.

1.3 Workshop Structure

After a peer review process where each submitted paper was reviewed by members of the international program committee, the workshop accepted papers for presentation and inclusion in the workshop proceedings. The workshop was organized in four sessions: testing, mobile and web, delta-oriented programming, and modeling.

2. OPEN RESEARCH TOPICS

After the paper presentations, discussions led to the following themes as topics for future work: modeling variability, testing, and usability and scalability as cross-cutting concerns.

2.1 Modeling Diverse Types of Variability

Modeling is about the "conceptual dimension" of variability. Nowadays, variability is considered in multiple dimensions and domains. While traditional modeling approaches focus on modeling the variability in features, functionalities and components of a software product line, more recently many authors have been focusing on the variability in other domains, such as the context or in the platform where the applications run. The existence of these "multiple variabilities" introduces new challenges for modeling.

On the one hand, the modelers must select which representations to use. Variability can be specified using external models such as *feature models* [11] and *orthogonal variability models* [13], using extensions to existing models such as UML stereotypes [8] or directly in the code such as using Java annotations, aspects or conditional compilation. Different representation schemas (such as aspect-oriented modeling [e.g., in AspectJ], delta-oriented modeling [e.g., DeltaJ], different architectural or system views and annotations in models) have been proposed. Although several authors propose to use only one type of representation, using a combination may be more appropriate.

On the other hand, modeling must maintain the consistency across all the representations and across levels of abstraction, from specification to implementation. When the variabilities in different domains are represented in multiple ways, it is desirable to define relationships among these representations and automated processes to determine interactions, inconsistencies and errors. There are some works aimed to check the consistency of the variability represented in models with the represented in code, e.g., cross-checking feature models and conditional compilation using #ifdefs. However, few works relate these variabilities with the existing in non-traditional domains such as the context or the platform.

As an example of these new challenges, consider the modeling of the context, i.e., the modeling of the external elements for a line of products. The number of surrounding elements and the number of combinations of these elements may exceed the number of selectable features for the line. A modeler may find difficulties not only to determine which of these elements worth to be modeled, but also which rules describe which combinations are valid. Representations such as feature models, where all the features and valid combinations must be fully identified, may be hard to use. Other types of representations, such as *probabilistic feature models* [2] may be considered.

As another example, consider the issues related to the *Android Fragmentation* [12]. There are many different versions of the Android OS, different types of devices, manufacturer skins and extensions, so that it is practically impossible to test a software product for all possible combinations. Someone interested on modeling the Android's platform variability must consider which types of models and validation techniques are the most suitable. For instance, although a probabilistic feature model may represent the features and combinations of features with greater probability of occurrence in a geographical region or market, automated processing of these models may require different techniques to the used for the traditional feature models.

2.2 Testing of Variability-intensive Systems

Testing is about the "quality dimension" of variability. The activities for quality assurance of variability-intensive systems must consider the diverse types of variabilities. For instance, existing approaches for software product lines aim to determine which set of combinations of features or components to test in order to obtain a pair-wise or a t-wise coverage. However, these approaches usually rely on processing a single representation of variability, e.g., a single feature model, and do not consider multiple representations or multiple types of variabilities. A key concern regarding testing of configurable systems is efficiency; besides collective offline/online testing, interesting approaches to be considered in this context are incremental testing and identifying equivalence classes of configurations to be tested.

In Dynamic Software Product Lines (DSPLs), the behavior of the system may vary not only depending of the features configured in the product but also on the elements detected in the context. To test one of these systems, it is necessary to determine variations in both, their inner features or components and in the surrounding elements of that system. In addition, some tests may involve changes in these elements in the context during an operation. Assuring the quality of DSPLs may require processing multiple representations of variability and diverse techniques for testing.

Novel techniques for testing DSPLs and variability-intensive systems must be explored. On the one hand, the techniques used to determine which variants to test can consider the variability observed in the reality. Instead of considering coverage such as pair-wise or t-wise combinations, the testing procedures can consider the combinations in the features of the system or in the context that are most common [9]. For instance, to test a mobile application, a novel technique may prioritize the platforms and the elements in the context with more probability of occurrence in a region or market. Techniques to capture the most common variability and determine an *observation space* must be developed. On the other hand, part of the testing can be performed at runtime when a concrete non-tested combination of context, platform and features is detected. Testing can combine *off-line testing* where a more-exhaustive test is performed using combinations determined upfront, with *on-line testing* performed using combinations detected at runtime [9].

Note that the variability observed in a market, detected in the environment or imposed by the multiple platforms may exhibit a combinatorial explosion. Trying to test all the combinations is likely to be infeasible. It is important to reduce the number of combinations

considering similarities and equivalence on multiple types and representations of variability.

2.3 Usability and Scalability of Variability Models

Usability and scalability are cross-cutting concerns when it comes to variability handling and management approaches (e.g., modeling techniques, processes, practices and tools). Chen and Babar discuss several principles, mechanisms, and techniques proposed in the literature for achieving scalability when modeling variability [1]. The authors describe different kinds of "divide and conquer approaches", such as decomposition/composition or separation of concerns. They also discuss strategies for hiding unnecessary information at each time, or querying approaches for accessing only the relevant part of a large and complex model.

Although these approaches have been proposed having in mind the scalability of the variability models themselves, such techniques can aid making testing approaches for variability-intensive systems more scalable. In particular, since most of these approaches intend to manipulate just a part or a view of the model at a time, such approaches could be adopted to more easily to design *unit testing* approaches for the models. Indeed, given smaller parts of the models, to test each one can probably be easier than designing tests for complete, potentially too big, variability models. This deserves further investigation.

We believe the strategies presented in [1] can also be used to increase the usability of the modelling approaches, and of the testing ones too. Usability can be defined as the "extent to which a product can be used by specified users to achieve specified goals effectively, efficiently and with satisfaction in a specified context of use" [10]. In this case, the users are the modelers or the testers, and their goals are to define the models or the tests, respectively. Although arguable, to define smaller models and tests are tasks these users can perform more efficiently and effectively, thus raising their satisfaction. We argue these ideas deserve further studies to investigate their real impact in current modelling and testing techniques' usability.

3. CONCLUSIONS

We summarized the outcome of the Second International Workshop on Variability and Complexity in Software Design. We gave an overview of the event, summarized discussions and offered an outlook on themes that emerged from the discussions at the workshop and which might be subject to future work. Key themes for further investigation are modeling variability, testing variability intensive systems, and usability and scalability as cross-cutting concerns.

4. ACKNOWLEDGMENTS

The VACE workshop is a collective endeavor. The organizers would like to thank all workshop authors, presenters and submitters. We also thank the ICSE 2017 organizers and in particular the workshop chairs. Finally, we thank the members of the program committee.

5. REFERENCES

[1] Chen, L., and Babar, M. A. 2009. A survey of scalability aspects of variability modeling approaches. In *Workshop on Scalable Modeling Techniques for Software Product Lines at SPLC*'09.

[2] Czarnecki, K. She, S. and Wasowski, A. 2008. Sample spaces and feature models: There and back again. In *Software Product Line Conference SPLC'08*.

[3] Galster, M., Avgeriou, P., Weyns, D., and Mannisto, T. 2011. Variability in Software Architecture: Current Practices and Challenges. *ACM SIGSOFT Software Engineering Notes* 36, 5 (2011), 30-32.

[4] Galster, M., Mannisto, T., Weyns, D., and Avgeriou, P. 2014. Variability in Software Architecture: The Road Ahead. *ACM SIGSOFT Software Engineering Notes* 39, 4 (2014), 33-34.

[5] Galster, M., Weyns, D., Tofan, D., Michalik, B., and Avgeriou, P., Variability in Software Systems - A Systematic Literature Review, *IEEE Transactions on Software Engineering* 40, 3 (2014), 282-306.

[6] Galster, M., Weyns, D., Avgeriou, P., and Becker, M. 2013. Variability in Software Architecture: Views and Beyond. *ACM SIGSOFT Software Engineering Notes* 38, 1 (2013), 46-49.

[7] Galster, M., Zdun U., Weyns, D., Rabiser, R., Zhang, B., Goedicke, M., and Perrouin, G. 2016. Variability and Complexity in Software Design: Towards a Research Agenda. *ACM SIGSOFT Software Engineering Notes* 41, 6 (2016), 27-30.

[8] Gomaa, H. 2004. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, Addison Wesley.

[9] Hansel, J and Giese, H. 2017. Towards Collective Online and Offline Testing for Dynamic Software Product Line Systems. In *IEEE/ACM 2nd International Workshop on Variability and Complexity in Software Design*.

[10] ISO/IEC. 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11 Guidance on usability, ISO/IEC 9241-11: 1998 (en), 1998.

[11] Kang, K., Cohen, S. Hess, J. 1990. Feature-oriented domain analysis (FODA) feasibility study. Technical Report. Software Engineering Institute.

[12] OpenSignal. 2015. Android Fragmentation Report August 2015. https://opensignal.com/reports/2015/08/android-fragmentation/

[13] Pohl, K. Bockle, G. Van der Linden, F. 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer.